Attorney Docket No.: 16869Q-106000US
Client Reference No.: HSJ8-2003-0361

PATENT APPLICATION

TECHNIQUES FOR DETECTING AND CORRECTING ERRORS USING MULTIPLE INTERLEAVE ERASURE POINTERS

Inventor(s): Martin Hassner, a citizen of The United States, residing at
1229 Christobal Privada
Mountain View, CA 94040

Vipul Srivastava, a citizen of India, residing at
4300 The Woods Dr., #1623
San Jose, CA 95136

Assignee: Hitachi Global Storage Technologies Inc.
5600 Cottle Road, NHGB / 014-2
San Jose, CA 95193

Entity: Large

TOWNSEND and TOWNSEND and CREW LLP
Two Embarcadero Center, Eighth Floor
San Francisco, California 94111-3834
Tel: 650-326-2400

PATENT

Attorney Docket No.: 16869Q-106000US
Client Reference No.: HSJ8-2003-0361

TECHNIQUES FOR DETECTING AND CORRECTING ERRORS USING MULTIPLE INTERLEAVE ERASURE POINTERS

BACKGROUND OF THE INVENTION

[0001] The present invention relates to techniques for detecting and correcting errors on storage media, and more particularly, to techniques for detecting and correcting errors on storage media using multiple interleave erasure pointers in a multi-level integrated interleave format.

[0002] Data bytes are stored on magnetic media in groups referred to as codewords (sectors). Typically, 512 data bytes are stored in each codeword. The codewords of data bytes may include multiple interleaved words (i.e., data byte strings or vectors) and corresponding check bytes that are byte interleaved.

[0003] Multibyte errors are detected and corrected in long byte strings recorded on a storage medium in blocks. Each block comprises a plurality of codewords and a plurality of block-level check bytes derived from the codewords. Each codeword includes data bytes and codeword check bytes mapped from a plurality of equal-length data byte strings according to a linear error correction code.

[0004] Each block is generated and recorded by logically summing the data byte strings and by mapping the logical sum and the data byte strings into counterpart codewords. The counter part codewords include codeword check bytes in accordance with the same linear error correction code. Next, the codewords are logically summed. The codewords and their logical sum are interleaved in a predetermined pattern prior to being recorded on a storage device or the like.

[0005] Additional details of a technique for detecting and correcting error bytes is discussed in U.S. Patent 5,946,328, which is incorporated by reference herein. Three data streams $m_1(x)$, $m_2(x)$, and $m_3(x)$ are used as an example. Two of the data streams $m_1(x)$ and $m_2(x)$ are encoded by a first Reed-Solomon (RS) linear encoder that produces respective codewords $c_1(x)$ and $c_2(x)$. This first RS encoder appends $2t_1$ check bytes to each of the codewords.

[0006] The third datastream $m_3(x)$ is modified to form the logical (modulo 2) sum of $m_1(x)+m_2(x)+m_3(x)$ prior to encoding by a second RS encoder. This second RS encoder includes

$2t_1 + 2t_2$ check bytes within a codeword $c'(x)$ of the logically-summed datastream. The codeword $c_3(x)$ represents the logical sum of the three codewords $c_1(x)+c_2(x)+c'(x)$. The third codeword $c_3(x)$ contains $2t_2$ shared block check bytes and $2t_1$ individual check bytes when generated in this manner. The block check bytes are inside and an intrinsic part of the RS codeword.

[0007] Two of the codewords $c_1(x)$ and $c_2(x)$ are generated by linear error correction encoding of respective data byte strings $m_1(x)$ and $m_2(x)$. Codewords $c_1(x)$ and $c_2(x)$ require $2t_1$ check bytes. The codeword $c'(x)$ are also generated by the linear encoding of the modulo 2 sum of $m_1(x)$, $m_2(x)$, and $m_3(x)$. Codeword $c'(x)$ requires $2t_1 + 2t_2$ check bytes. Data byte strings $m_1(x)$ and $m_2(x)$ are appended with $2t_2$ zeroes denoted by $\varphi(x)$ in order to secure equal codeword length. The codeword outputs may then be expressed as:

$$c_1(x) = x^{2t1+2t2}\, m_1(x) + \varphi(x) + r_1(x)$$

$$c_2(x) = x^{2t1+2t2}\, m_2(x) + \varphi(x) + r_2(x)$$

$$c'(x) = x^{2t1+2t2}[m_1(x) + m_2(x) + m_3(x)] + r(x)$$

[0008] The codeword $c'(x)$ is further processed to produce a modified and third codeword $c_3(x)$ by summing the three codewords $c_1(x)$, $c_2(x)$, and $c'(x)$ modulo 2 such that:

$$c_3(x) = [c_1(x) + c_2(x)] + c'(x) = x^{2t1+2t2}\, m_3(x) + r_B(x) + r_3(x)$$

[0009] The check bytes $r_B(x)$ are the block check bytes shared by $m_1(x)$, $m_2(x)$, and $m_3(x)$, whereas $r_3(x)$ are the individual check bytes of data stream $m_3(x)$.

[0010] The check bytes $r_1(x)$, $r_2(x)$, and $r_3(x)$ are first level check bytes. These check bytes can only be used to correct t_1 errors in data bytes within codewords $c_1(x)$, $c_2(x)$, $c_3(x)$, respectively. The check bytes $r_B(x)$ in codeword $c_3(x)$ are second (block) level check bytes that can be used to correct $t_1 + t_2$ data byte errors in any one of codewords $c_1(x)$, $c_2(x)$, and $c_3(x)$.

[0011] Following these computations, an integrated interleaved block of codewords $c_1(x)$, $c_2(x)$, and $c_3(x)$ is written out to the disk. Subsequently, when the disk must execute a read or read modify write command or the like, an addressed block or blocks of codewords are streamed from their track locations on the disk. Any errors on the disk are detected and corrected on the fly based on the syndrome processing of the codewords. The syndromes are derived from the purported codewords and their logical sum modulo 2. A non-zero syndrome indicates an error byte. Any nonzero syndromes are identified and processed over the codewords to correct the bytes in error. The updated block-level nonzero syndromes are also processed to locate and correct bytes in error in any single codeword that exceeds the correction capability of that codeword.

[0012] U.S. Patent 6,275,965, which is incorporated herein by reference, describes a system that corrects error bytes in a two-level code structure. Up to a maximum of t_1 error bytes in a codeword can be located and corrected using $2t_1$ check bytes per codeword. A codeword with error bytes exceeding its t_1 codeword correction capacity is a bursty subblock. Up to a maximum of $t_1 + t_2$ error bytes can be located and corrected in as many as B bursty subblocks using $B^*(2t_2)$ block-level check bytes.

[0013] A post-encoding process is provided to "de-interleave" or redistribute the computed codewords into modified codewords such that the data vectors and their codeword check bytes are located in the same codeword. The block-level check bytes are distributed among the codewords.

[0014] The techniques described in the '965 patent are useful for correcting error bytes that occur in random locations with the codewords. However, the number of error bytes in the codewords can exceed the correction capability of the techniques described in the '965 patent. These types of errors are referred to as burst errors. Burst errors can occur, for example, as the result of a scratch or defect on a magnetic disk.

[0015] Therefore, it would be desirable to provide additional techniques for detecting and correcting burst errors occurring in data bytes that are formed in a two-level block code structure.

BRIEF SUMMARY OF THE INVENTION

[0016] The present invention provides techniques for detecting and correcting burst errors in data bytes formed in a two-level block code structure. According to the present invention, a second level decoder uses block level check bytes to detect columns in a two-level block code structure that contain error bytes. The second level decoder generates erasure pointers that identify columns in the two-level block structure that have been effected by burst errors.

[0017] A first level decoder then uses codeword check bytes to correct all of the bytes in the columns identified by the erasure pointers. The present invention frees up the first level decoder to use all of the codeword check bytes only for error byte value calculations. The first level decoder does not need to use any of the codeword check bytes for error location calculations, because the erasure pointers generated by the second level decoder provide all of the necessary error locations. The error correction capability of the first level decoder is doubled, because all of the codeword check bytes are freed up to be used only for error value calculations.

[0018] Other objects, features, and advantages of the present invention will become apparent upon consideration of the following detailed description and the accompanying drawings, in which like reference designations represent like features throughout the figures.

BRIEF DESCRIPTION OF THE DRAWINGS

[0019] Figure 1A is a block diagram detailing the architecture of a buffered hard disk drive controller that includes an on-the-fly error correction code (ECC) system for implementing on-the-fly error correction code.

[0020] Figure 1B is a block diagram of a data storage system depicting data flow along a read channel and a write channel of the hard disk drive controller of Figure 1A.

[0021] Figure 2 illustrates a two-level block code structure according to the present invention.

[0022] Figure 3 illustrates a first level decoder for detecting and correcting random errors according to the present invention.

[0023] Figure 4 illustrates a second level decoder for detecting burst errors according to the techniques of the present invention.

[0024] Figure 5 illustrates a first level decoder for correcting burst errors according to the techniques of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

[0025] Figures 1A and 1B illustrate an example of a hard disk drive control system for reading and writing data onto a magnetic hard disk. The hard disk drive control system of Figures 1A-

1B is an example of hard disk drive system that can implement techniques of the present invention that are described above. The hard disk drive system of Figures 1A-1B detect and correct errors in the data read from a disk. The hard disk drive system of Figures 1A-1B can be used to implement the error correction techniques of the present invention, which are discussed in further detail below.

[0026] Figure 1A illustrates an exemplary architecture of a buffered hard disk drive controller 50. Hard disk drive controller 50 is configured to read data from and write data to a magnetic hard disk 14. Controller 50 includes an on-the-fly error correction code (ECC) system 100 for implementing an on-the-fly error correction code. On-the-fly error correction code system 100 includes an ECC read processor 163 and an ECC write processor 167.

[0027] Figure 1B is a block diagram of the hard disk drive controller 50 of Figure 1A that includes an on-the-fly error correction code system 100. When sequences of digital binary data are to be written onto the disk 14, they are placed temporarily in a buffer 165 shown in Figure 1A and subsequently processed and transduced along a write path or channel (167, 169, and 157).

[0028] The hard disk drive controller 50 includes a logic drive circuit 105 shown in Figure 1B that formats data from hard disk assembly 33, for example from 8 bits to 32 bits. A disk assembly 33 is comprised of disk 14 and a head stack assembly including a spindle motor. A FIFO register 110 stores the formatted data and exchanges the same with a sector buffer 120. The ECC system 100 receives the formatted data from the drive logic circuit 105 and performs an error correction coding algorithm. ECC system 100 can also perform the techniques and processes of the present invention, which are discussed in detail below.

[0029] A buffer manager 115 controls data traffic between the ECC system 100, a sector buffer (i.e., random access memory) 120, and a microprocessor 125. Another FIFO register 130 stores data and exchanges the same with the sector buffer 120. A sequence controller 135 is connected between the drive logic circuit 105, the microprocessor 125, and a host interface 140, to control the sequence operation of the data traffic and various commands across the hard drive controller 50. The host interface 140 provides an interface between the hard drive controller 50 and a host 60.

[0030] First, a predetermined number of binary data elements, also termed bytes, in a data string are moved from the buffer 165 and streamed through an ECC write processor 167. In the ECC write processor 167, the data bytes are mapped into codewords drawn from a suitable linear block or cyclic code such as a Reed-Solomon code. Next, each codeword is mapped in a write path signal-shaping unit 169 into a run length limited or other bandpass or spectral-shaping code and changed into a time-varying signal. The time-varying signal is applied through an interface read/write transducer interface 157 and then to the write element in a magneto resistive (or other suitable transducer head) for conversion into magnetic flux patterns.

[0031] All of the measures starting from the movement of the binary data elements from buffer 165 until the magnetic flux patterns are written on a selected disk track as the rotating disk 14 passes under the read/write head are synchronous and streamed. For purposes of efficient data transfer, the data is de-staged (written out) or staged (read) a codeword at a time.

[0032] Thus, both the mapping of binary data into Reed-Solomon codewords and the conversion to flux producing time-varying signals must be done well within the time interval defining a unit of recording track length moving under the transducer. Typical units of recording track length are equal fixed-length byte codewords of 512 bytes.

[0033] When sequences of magnetic flux patterns are to be read from the disk 14, they are processed in a read path or channel (157, 159, 161, and 163) and written into the buffer 165. The time-varying signals sensed by a transducer are passed through the read/write transducer interface 157 to a digital signal extraction unit 159. Here, the signal is detected and a decision is made as to whether it should be resolved as a binary 1 or 0. As these 1's and 0's stream out of the signal extraction unit 159, they are arranged into codewords in the formatting unit 161.

[0034] Because the read path is evaluating sequences of Reed-Solomon codewords previously recorded on the disk 14, absent error or erasure, the codewords should be the same. In order to test whether that is the case, each codeword is applied to an ECC read processor 163 over a path from a formatter 161.

[0035] Also, the output from the ECC processor 163 is written into buffer 165. The read path also operates in a synchronous data-streaming manner such that any detected errors must be located and corrected within the codeword well in time for the ECC read processor 163 to receive the next codeword read from the disk track. The buffer 165 and the read and write channels may be monitored and controlled by the microprocessor 125 to ensure efficacy where patterns of referencing may dictate that a path not be taken down, such as sequential read referencing.

[0036] Data bytes are typically stored in a data storage disk in codewords. Subsets of the codewords are grouped into long blocks. Each codeword contains first level CRC/ECC bytes, and each block includes second level ECC_B bytes.

[0037] A data storage system uses the first level CRC and ECC bytes to locate and to correct errors in data bytes within a codeword. When there are too many errors in the data bytes to be corrected by the first level ECC bytes, the data storage system uses the second level ECC_B bytes to correct the errors as described in detail above. ECC_B bytes are used for correcting data bytes in any of the codewords in a block. Miscorrection of an error can happen when both of the first level and the second level CRC and ECC bytes are in error.

[0038] Further details of multiple level, integrated sector format, error correction code, encoding and decoding processes for data storage or communication devices is discussed in further detail in U.S. Patent Application Publication US 2003/0147167 A1, published August 7, 2003, to Asano et al., which is incorporated by reference herein.

[0039] Burst errors (also called hard errors) occur when a scratch or defect on a storage medium such as a magnetic disk wipes out several adjacent data bytes. Because data bytes are stored next to each other on a magnetic disk in codewords and blocks of data bytes, burst errors wipe out data bytes that are located next to each other within the codewords of a block of data.

[0040] The present invention provides techniques for detecting and correcting data bytes that have been corrupted by burst errors on a storage medium. The techniques of the present invention take advantage of the fact that burst errors result in corrupted data bytes that are located next to each other within codewords of a block.

[0041] A plurality of data streams are encoded to generate codewords as discussed above and in further detail in U.S. Patents 5,946,328 and 6,275,965. The codewords include data bytes and check bytes. The codewords include two levels of check bytes. Each codeword $c_1, c_2, c_3, \ldots$ includes a first level check byte $r_1, r_2, r_3, \ldots$, respectively, that can be used to correct t_1 data bytes with errors. One or more of the codewords also include second (block) level check bytes r_B that can be used to correct $t_1 + t_2$ data bytes in any of the codewords that have errors.

[0042] The codewords that contain only the first level check bytes are referred to as the first level codewords. The codewords that contain the second level check bytes are referred to as the second level codewords.

[0043] In an interleaved format, the first and second level codewords are arranged in a two-level block code structure. The block can be expressed such that each codeword $c_1, c_2, c_3 \ldots$ is a row in the block as shown in Figure 2. The first level codewords are nested to form a first code C1 in the block, and the second level codewords are nested to form a second code C2 in the block.

[0044] Each codeword has n total bytes. The number of data bytes in each first level codeword is k1. The number of data bytes in each second level codeword is k2. Each of the first level codewords has $2t_1$ check bytes, and each of the second level codewords has $2t_2$ check bytes. B is the number of bursty codewords that are correctable, and m is the number of data vectors.

[0045] The Hamming distance between two codewords is the number of symbols in which they disagree. It can be shown that to correct T byte errors, a coding scheme requires the use of codewords with a Hamming distance of at least $2T + 1$ between the codewords. The Hamming distance of the first code C1 is $d_1 = 2t_1 + 1$. The Hamming distance of the second code C2 is $d_2 = 2t_2 + 1$, where $d_1 < d_2$.

[0046] The block C shown in Figure 2 includes codes C1 and C2. Block C is applied to the inverse of an integrating matrix V^{-1} to redistribute the data vectors, the codeword check bytes, and the block-level check bytes, as discussed in further detail with respect to Figures 5 and 6 of U.S. Patent 6,275,965. The integrating matrix is shown in Figure 2.

[0047] A decoder is used to generate the error location and correction values from the syndromes detected in any of the codewords that form an interleaved block. Before decoding a block, the block is first de-multiplexed to separate out the data byte strings, block check bytes, and the codeword check bytes so that they can be validity tested. The syndrome generators generate syndromes for the codewords in the block. Syndrome generating techniques are well-known to those of skill in the art.

[0048] The syndromes derived from the codewords are applied to a first level Reed-Solomon decoder. The first level Reed-Solomon decoder can detect and correct up to τ_1 bytes in error in any single codeword. Reed-Solomon decoders are well known to those skilled in the art.

[0049] Figure 3 illustrates an example of a block 200 that contains four codewords c_1, c_2, c_3, and c_4. Each codeword includes a data vector m and check bytes (the shaded portion). There are four data vectors m in block 200 and B = 1. The codewords c_i are applied to a first level decoder 201.

[0050] Decoder 201 can detect and correct random errors occurring in data vectors in the codewords. Decoder 201 uses syndromes to detect and correct up to τ_1 bytes in error in any one of the four codewords. Decoder 201 outputs corrected codewords c'_i. The corrected codewords include the corrected data vectors. The number of errors that are corrected τ_i is less than half the first level Hamming distance ($\tau_i < d_1/2$).

[0051] Another syndrome generator produces any nonzero syndromes based on the second (block) level check bytes in the block. These syndromes are modified to remove the effects of any errors located by the first level decoder. If the updated syndromes are zero, the correction computed by the first level decoder is deemed correct. If, however, the updated syndromes are nonzero or if the first level decoder detects a failure, then the first level syndromes in the phase that failed and the second level block syndromes are applied to a second level Reed-Solomon decoder.

[0052] Thus, if the first level decoder 201 is unsuccessful at correcting the errors in one of the codewords, the second level Reed-Solomon decoder attempts to correct the bytes in error. A second level Reed-Solomon decoder uses the codeword check bytes and the block check bytes to correct bytes in error as shown in Figure 4. The second level decoder can correct up to τ_2 bytes in error in the sum of all the codewords.

[0053] The two level block structure is such that the sum of all of the interleaves is a codeword in the second code C2. In the presence of a burst error, adjacent symbols in a codeword are erased. The symbols in the second level codeword, obtained as the sum of all the interleaves, are sums of adjacent symbols in the codeword. Thus, error locations computed by the second level decoder are natural erasure locations for all of the interleaves. The present invention takes advantage of this fact to locate and correct burst errors.

[0054] The codewords c_i are applied to a second level decoder 301 shown in Figure 4. Second level decoder 301 can locate columns in a two-level block code structure that have been affected by burst errors. Decoder 301 receives syndromes generated from the codeword check bytes and syndromes generated from the block level check bytes. Using these syndromes, decoder 301 can detect the columns that contains errors.

[0055] Second level decoder 301 outputs erasure pointers E_i. The erasure pointers point to columns of bytes in the two-level block structure that contain errors. Thus, the second level code C2 is only used to detect the locations of error bytes in the block. Because burst errors affect adjacent bytes, typically an entire column of bytes in a block is wiped out. The second level code C2 (that includes the block level check bytes) identifies the columns that have the errors. In a burst error situation, it is likely that many or all of the bytes in an effected column contain errors. The first level code C1 is then used to correct all of the bytes in the columns identified by the erasure pointers.

[0056] For example, block 300 in Figure 4 contains four codewords c_1, c_2, c_3, and c_4. Erasure pointers 311, 312, and 313 point to the first, third, and fourth byte columns in block 300. The number of errors that can be corrected τ_2 is less than half the second level Hamming distance ($\tau_2 < d_2/2$).

[0057] The erasure pointers E_i and the codewords c_i are then inputted into a first level decoder 401 as shown in Figure 5. First level decoder 401 corrects burst errors in a two level block code structure that has been previously processed by second level decoder 301. Decoder 401 interprets the erasure pointers generated by second level decoder 301 as locations of the bytes in error. Each erasure pointer points to a block column that contains error bytes. The first level decoder 401 uses the first level codeword check bytes to correct each of the bytes in the columns identified by the erasure pointers. The first level decoder 401 outputs corrected codewords c'_i. If the number of erasure pointers E_i equals ρ_i, then all possible combinations of τ_i and ρ_i are true such as that $2\tau_i + \rho_i < d_1$.

[0058] According to the present invention, the second level decoder 301 only uses the block level check bytes to detect the columns that contain error bytes. The block level check bytes are not used for error correction. The first level decoder 401 then uses the codeword check bytes and the erasure pointers to correct all of the bytes in the identified columns.

[0059] This technique frees up the first level decoder to use the codeword check bytes only for error value calculations, because the first level decoder does not need to use any of the codeword check bytes for error location calculations. The error correction capability of the first level decoder doubles, because all of the codeword check bytes are freed up to be used for error byte value calculations. Prior art systems used the first level check bytes for error detection and correction.

[0060] As discussed above, the prior art techniques described in U.S. Patents 5,946,328 and 6,275,965 are useful for correcting random errors in data streams organized in a two-level block structure. The system may not be able to determine if the errors are random errors or burst errors. Therefore, if the prior art techniques fail to correct all of the bytes in error, the techniques of the present invention can be attempted to correct the bytes in error. If the errors are burst errors, the present invention is more likely to be able to correct all of the bytes in error.

[0061] While the present invention has been described herein with reference to particular embodiments thereof, a latitude of modification, various changes, and substitutions are intended in the present invention. In some instances, features of the invention can be employed without a corresponding use of other features, without departing from the scope of the invention as set forth. Therefore, many modifications may be made to adapt a particular configuration or method disclosed, without departing from the essential scope and spirit of the present invention. It is intended that the invention not be limited to the particular embodiment disclosed, but that the invention will include all embodiments and equivalents falling within the scope of the claims.